UNIT IV
MINING DATA STREAMS

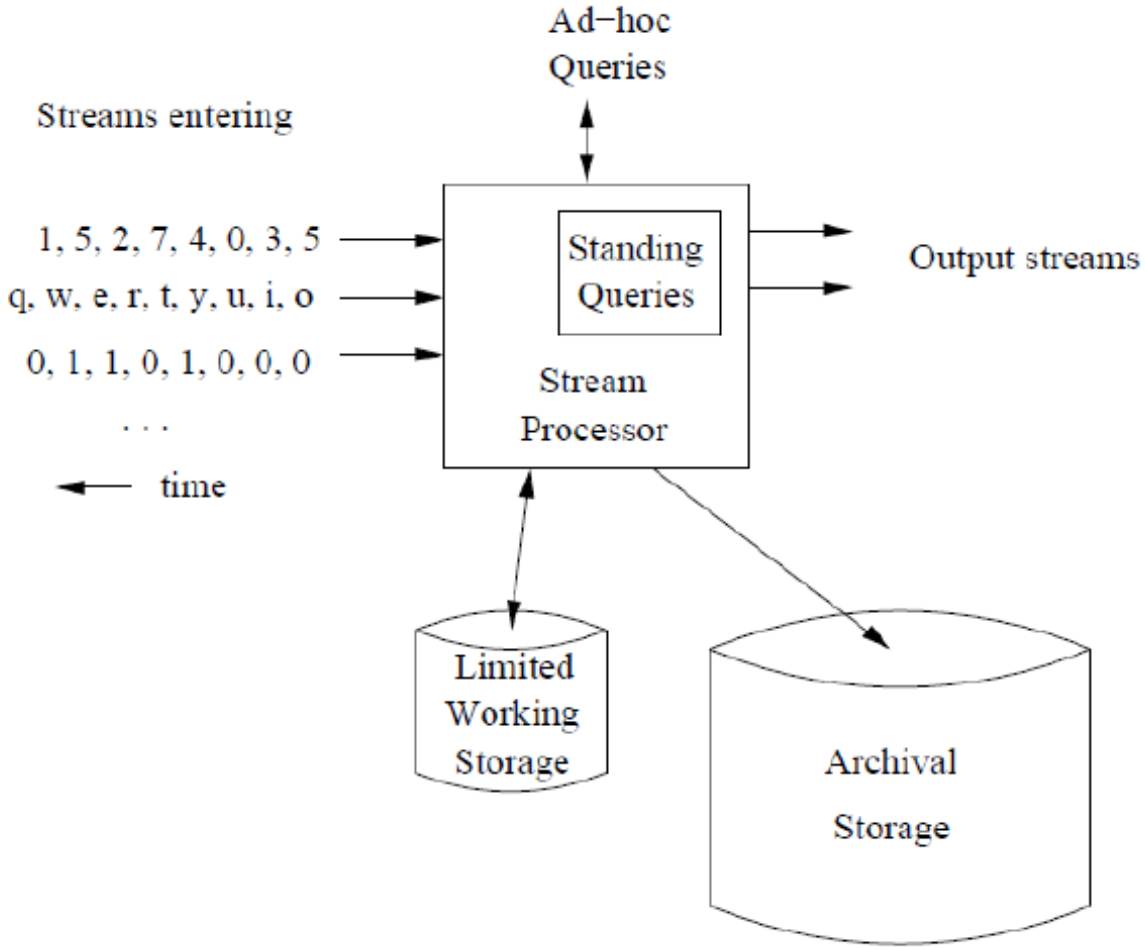
4.1. STREAMS: CONCEPTS

Introduction to streams concepts

Data streams are data arrives in a stream or streams, and if it is not processed immediately or stored, then it is lost forever. It is assume that the data arrives so rapidly that it is not feasible to store it in active storage (conventional database), and then interact with it at the time of our choosing.

Data Stream Model

The stream processor as a kind of data-management system which has the high-level organization as shown in the figure. Any number of streams can enter the system. Each stream can provide elements at its own schedule; they need not have the same data rates or data types, and the time between elements of one stream need not be uniform. The rate of arrival of stream elements is not under the control of the system. Later, the system controls the rate at which data is read from the disk, and therefore never data getting lost as it attempts to execute queries.



Streams may be archived in a large archival store, but we assume it is not possible to answer queries from the archival store. It could be examined only under special circumstances using time-consuming retrieval processes. There is also a working store, into which summaries or parts of streams may be placed, and which can be used for answering queries. The working store might be disk, main memory, depending on how fast we need to process queries. But either way, it is of sufficiently limited capacity that it cannot store all the data from all the streams.

Sources of Data Streams

Let us consider some of the ways in which stream data arises naturally.

Sensor Data

Sensor data like temperature, wind speed, climate, ocean behaviour or GPS information arriving every day, every hour, it need to think about what can be kept in working storage and what can only be archived. Such data can be collected by deploying thousands and thousands of sensors, each sending back a stream.

Image Data

Satellites often send down to earth streams consisting of many terabytes of images per day. Surveillance cameras produce images with lower resolution than satellites, but there can be many of them, each producing a stream of images at intervals like one second.

Internet and Web Traffic

Switching node in the Internet receives streams of IP packets from many inputs and routes them to its outputs. Normally, the role of the switch is to transmit data and not to retain it or query it. But there is a tendency to put more capability into the switch, e.g., the ability to detect denial-of-service attacks or the ability to reroute packets based on information about congestion in the network.

Web sites receive streams of various types For example; an increase in queries like “sore throat” enables us to track the spread of viruses. A sudden increase in the click rate for a link could indicate some news connected to that page, or it could mean that the link is broken and needs to be repaired.

Stream Queries

Queries may be answered for stream data by many ways. Some of them mat require average of specific number of elements or the one it may be maximum value. We might have a standing query that, each time a new reading arrives, produces the average of the 24 most recent readings. That query also can be answered easily, if we store the 24 most recent stream elements. When a new stream element arrives, we can drop from the working store the 25th most recent element, since it will never again be needed.

The other form of query is ad-hoc, a question asked once about the current state of a stream or streams. If we do not store all streams in their entirety, as normally we can not, then we cannot expect to answer arbitrary queries about streams. If we have some idea what kind of queries will be asked through the ad-hoc query interface, then we can prepare for them by storing appropriate parts or summaries of streams.

A common approach is to store a sliding window of each stream in the working store. A sliding window can be the most recent n elements of a stream, for some n, or it can be all the elements that arrived within the last t time units, If we regard each stream element as a tuple, we can treat the window as a relation and query it with any SQL query.

Issues in Data streaming

Streams often deliver elements very rapidly. So it is required to process elements in real time, or lose the opportunity to process them at all, without accessing the archival storage. Thus, it often is important that the stream-processing algorithm is executed in main memory without access to secondary storage.

4.2. STREAM DATA MODEL AND ARCHITECTURE

Introduction

Traditional DBMSs have been oriented toward business data processing, and consequently are designed to address the needs of these applications.First, they have assumed that the DBMS is a passive repository storing a large collection of data elements and that humans initiate queries and transactions on this repository.W e call this a *human-active, DBMS-passive (HADP)* model.Second,

they have assumed that the current state of the data is the only thing that is important.Hence, current values of data elements are easy to obtain, while previous values can only be found torturously by decoding the DBMS log.The third assumption is that triggers and alerters are second-class citizens.These constructs have been added as an afterthought to current systems, and none has an implementation that scales to a large number of triggers.F ourth, DBMSs assume that data elements are synchronized and that queries have exact answers.In many stream-oriented applications, data arrive asynchronously and answers must be computed with incomplete information.Lastly , DBMSs assume that applications require no real-time services.There is a substantial class of applications where all five assumptions are problematic.Monitoring applications are applications that monitor continuous streams of data.This class of applications includes military applications that monitorreadings from sensors worn by soldiers (e.g., blood pressure, heart rate, position), financial analysis applications that monitor streams of stock data reported from various stock exchanges, and tracking applications that monitor the locations of large numbers of objects for which they are responsible (e.g., audiovisual departments that must monitor the location of borrowed equipment).Because of the high volume of monitored data and the query requirements for these applications, monitoring applications would benefit from DBMS support.Existing DBMS systems, however, are ill suited for such applications since they target business applications. First, monitoring applications get their data from external sources (e.g., sensors) rather than from humans issuing transactions.The role of the DBMS in this context is to alert humans when abnormal activity is detected.This is a *DBMSactive, human-passive (DAHP)* model. Second, monitoring applications require data management that extends over some history of values reported in a stream and not just over the most recently reported values. Consider a monitoring application that tracks the location of items of interest, such as overhead transparency projectors and laptop computers, using electronic property stickers attached to the objects. Ceiling-mounted sensors inside a building and the GPS system in the open air generate large volumes of location data.If a reserved overhead projector is not in its proper location,then one might want to know the geographic position of the missing projector.In this case, the last value of the monitored object is required. Ho wever, an administrator might alsowant to know the duty cycle of the projector, thereby requiring access to the entire historical time series.Third, most monitoring applications are trigger-oriented.If one is monitoring a chemical plant, then one wants to alert an operator if a sensor value gets too high or if another sensor value has recorded a value out of range more than twice in the 2 last 24 h.Ev ery application could potentially monitor multiple streams of data, requesting alerts if complicated conditions are met.Thus, the scale of trigger processing required in this environment far exceeds that found in traditional DBMS applications.

Fourth, stream data are often lost, stale, or intentionally omitted for processing reasons.An object being monitored may move out of range of a sensor system, thereby resulting in lost data.The most recent report on the location of the object becomes more and more inaccurate over time.Moreo ver, in managing data streams with high input rates, it might be necessary to shed load by dropping less important input data. All of this, by necessity, leads to approximate answers.

Lastly, many monitoring applications have real-time requirements. Applications that monitor mobile sensors (e.g., military applications monitoring soldier locations) often have a low tolerance for stale data, making these applications effectively real time.The added stress on a DBMS that must serve real-time applications makes it imperative that the DBMS employ intelligent resource management (e.g., scheduling) and graceful degradation strategies (e.g., load shedding) during periods of high load.W e expect that applications will supply quality-of-service (QoS) specifications that will be used by the running system to make these dynamic resource allocation decisions.

Monitoring applications are very difficult to implement in traditional DBMSs.First, the basic computation model is wrong: DBMSs have a HADP model while monitoring applications often require a DAHP model.In addition, to store time-series information one has only two choices.First, he can encode the time series as current data in normal tables.In this case, assembling the historical time series is very expensive because the required data is spread over many tuples, thereby dramatically slowing performance.Alternately, he can encode time series information in binary large

objects to achieve physical locality, at the expense of making queries to individual values in the time series very difficult.One system that tries to do something more intelligent with time series data is the Informix Universal Server, which implemented a time-series data type and associated methods that speed retrieval of values in a time series however, this system does not address the concerns raised above.

If a monitoring application had a very large number of triggers or alerters, then current DBMSs would fail because they do not scale past a few triggers per table.The only alternative is to encode triggers in some middleware application.Using this implementation, the system cannot reason about the triggers (e.g., optimization), because they are outside the DBMS. Moreover, performance is typically poor because middleware must poll for data values that triggers and alerters depend on. Lastly, noDBMSthat we are aware of has built-in facilities for approximate query answering.The same comment applies to real-time capabilities.Again, the user must build custom code into his application. For these reasons, monitoring applications are difficult to implement using traditional DBMS technology.T o do better, all the basic mechanisms in currentDBMSsmust be rethought.

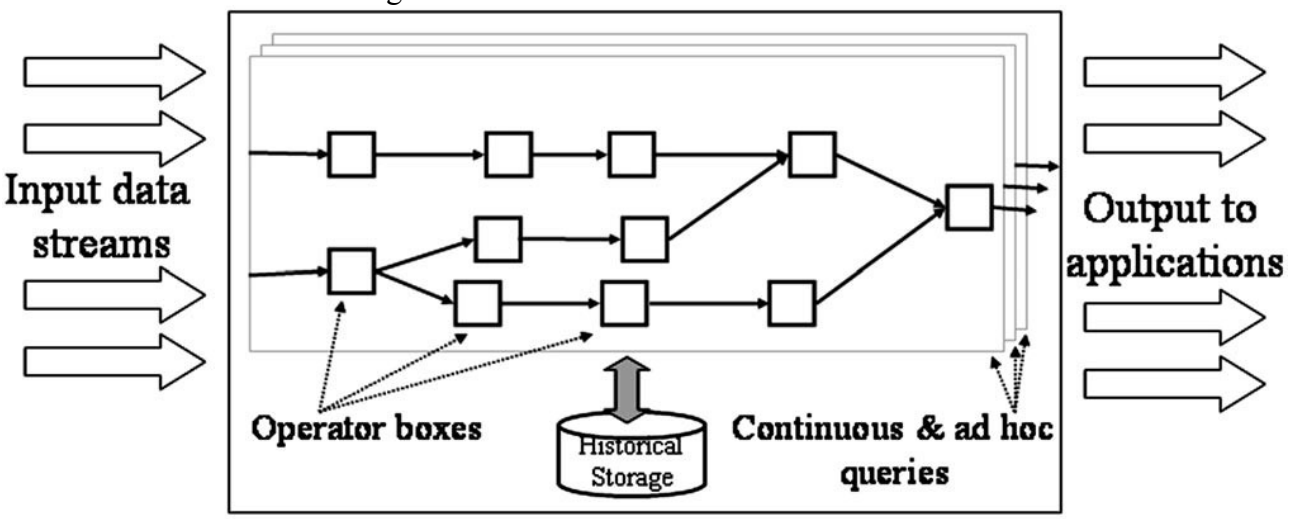


Fig. 1. Aurora system model

Monitoring applications are applications for which streams of information, triggers, imprecise data, and real-time requirements are prevalent.We expect that there will be a large class of such applications.F or example, we expect the class of monitoring applications for physical facilities (e.g., monitoring unusual events at nuclear power plants) to grow in response to growing needs for security.In addition, as GPSstyle devices are attached to an ever broader class of objects, monitoring applications will expand in scope.Currently such monitoring is expensive and restricted to costly items like automobiles (e.g., Lojack technology).

Aurora system model

Aurora data are assumed to come from a variety of data sources such as computer programs that generate values at regular or irregular intervals or hardware *sensors*.We will use the term *data source* for either case.In addition, a data *stream* is the term we will use for the collection of data values presented by a data source.Each data source is assumed to have a unique source identifier, andAurora timestamps every incoming tuple to monitor the quality of service being provided. The basic job of Aurora is to process incoming streams in the way defined by an *application administrator.*Aurora is fundamentally a data-flow system and uses the popular *boxes and arrows* paradigm found in most process flow and workflow systems.Hence, tuples flow through a loop-free, directed graph of processing operations (i.e., boxes). Ultimately, output streams are presented to *applications*, which must be programmed to deal with the asynchronous tuples in an output stream.Aurora can also maintain historical storage, primarily in order to support ad hoc queries.Figure 1 illustrates the high-level system model. Aurora's query algebra (SQuAl1) contains

built-in support for seven primitive operations for expressing its stream 1 SQuAl is short for [S]tream [Qu]ery [Al]gebra.

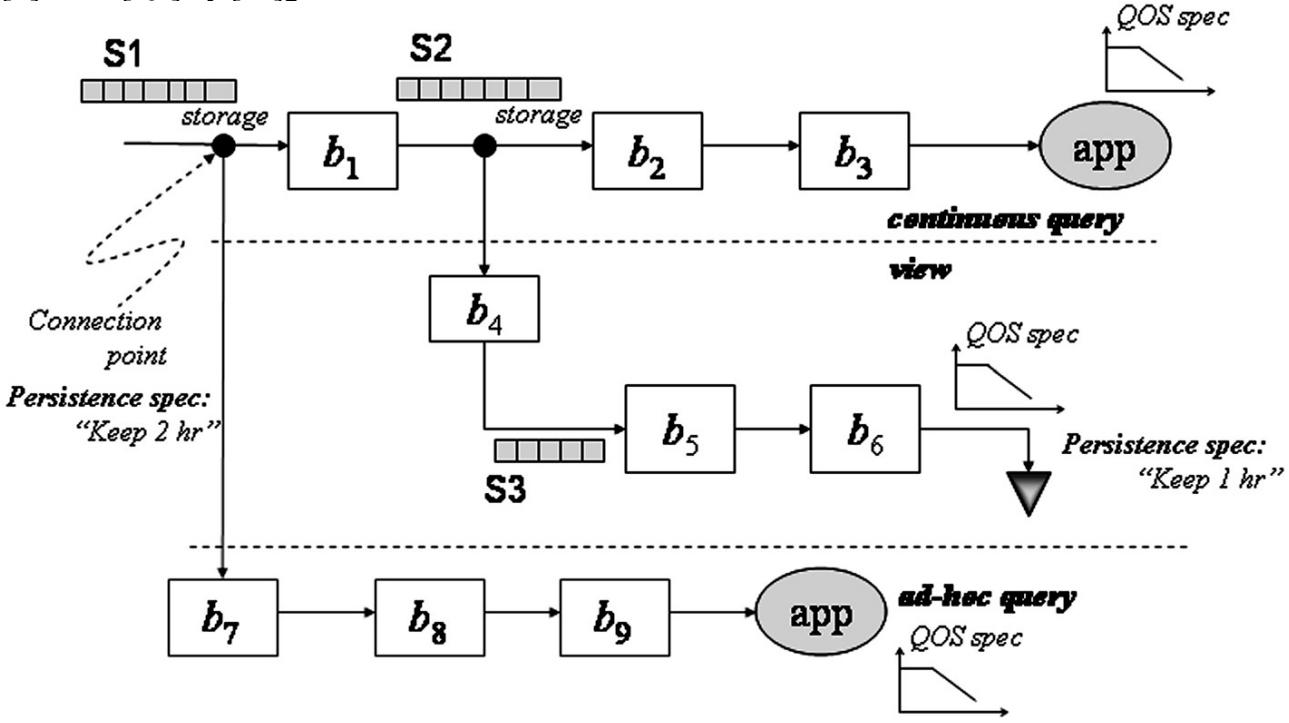


Fig. 2. Aurora query model

processing requirements.Man y of these have analogs in the relational query operators.F or example, we support a *filter* operator that, like the relational operator *select*, applies any number of predicates to each incoming tuple, routing the tuples according to which predicates they satisfy.Another operator, (*Aggregate*), computes stream aggregates in a way that addresses the fundamental push-based nature of streams, applying a function across a window of values in a stream (e.g., a moving average).In environments where data can be stale or time imprecise, windowed operations are a necessity. There is no explicit *split* box; instead, the application administrator can connect the output of one box to the input of several others.This implements an implicit *split* operation. On the other hand, there is an explicit Aurora *Union* operation, whereby two streams can be put together.If, additionally, one tuple must be delayed for the arrival of a second one, then a *Resample* box can be inserted in the Aurora network to accomplish this effect.

Arcs in an Aurora diagram actually represent a collection of streams with common schema.The actual number of streams on an arc is unspecified, making it easy to have streams appear and disappear without modification to the Aurora network.

Query model

Aurora supports continuous queries (real-time processing), views, and ad hoc queries all using substantially the same mechanisms.All three modes of operation use the same conceptual building blocks.Each mode processes flows based on *QoS specifications* – each output in Aurora is associated with two-dimensional QoS graphs that specify the utility of the output in terms of several performance-related and quality-related attributes (see Sect.4. 1). The diagram in Fig.2 illustrates the processing modes supported by Aurora. The topmost path represents a *continuous query*.In isolation, data elements flow into boxes, are processed, and flow further downstream.In this scenario, there is no need to store any data elements once they are processed. Once an input has worked its way through all reachable paths, that data item is drained from the network.The QoS specification at the end of the path controls how resources are allocated to the processing elements along the path.One can also view an Aurora network (along with some of its applications) as a large collection of triggers.Each path from a sensor input to an output can be viewed as computing the *condition* part of a complex trigger. An output tuple is delivered to an application, which can take

the appropriate action. The dark circles on the input arcs to boxes *b*1 and *b*2 represent *connection points*.A connection point is an arc that supports dynamic modification to the network.Ne w boxes can be added to or deleted from a connection point.When a new application connects to the network, it will often require access to the recent past.As such, a connection point has the potential for persistent storage .Persistent storage retains data items beyond their processing by a particular box.In otherwords, as items flowpast a connection point, they are cached in a persistent store for some period of time.They are not drained from the network by applications.Instead, a persistence specification indicates exactly how long the items are kept, so that a future ad hoc query can get historical results. In the figure, the leftmost connection point is specified to be available for *2 h*.This indicates that the beginning of time for newly connected applications will be 2 h in the past. Connection points can be generalized to allow an elegant way of including *static data sets* in Aurora.Hence we allow a connection point to have no upstream node, i.e., a dangling connection point.W ithout an upstream node the connection point cannot correspond to an Aurora stream.Instead, the connection point is decorated with the identity of a stored data set in a traditional DBMS or other storage system.In this case, the connection point can be *materialized* and the stored tuples passed as a stream to the downstream node.In this case, such tuples will be *pushed* through an Aurora network.Alternately, query execution on the downstream node can *pull* tuples by running a query to the store.If the downstream node is a filter or a join, pull processing has obvious advantages.Moreo ver, if the node is a join between a stream and a stored data set, then an obvious query execution strategy is to perform iterative substitution whenever a tuple from the stream arrives and perform a lookup to the stored data.In this case, a windowdoes not need to be specified as the entire join can be calculated.

The middle path in Fig.2 represents a view.In this case, a path is defined with no connected application.It is allowed to have a QoS specification as an indication of the importance of the view.Applications can connect to the end of this path whenever there is a need.Before this happens, the system can propagate some, all, or none of the values stored at the connection point in order to reduce latency for applications that connect later.Moreo ver, it can store these partial results at any point along a viewpath.This is analogous to a materialized or partially materialized view.V iew materialization is under the control of the scheduler.

The bottom path represents an ad hoc query.An ad hoc query can be attached to a connection point at any time.The semantics of an ad hoc query is that the system will process data items and deliver answers from the earliest time *T* (persistence specification) stored in the connection point until the query branch is explicitly disconnected.Thus, the semantics for an Aurora ad hoc query is the same as a continuous query that starts executing at *tnow* − *T* and continues until explicit termination.

Graphical user interface

The Aurora user interface cannot be covered in detail because of space limitations.Here, we mention only a few salient features. T o facilitate designing large networks, Aurora will support a hierarchical collection of groups of boxes.A designer can begin near the top of the hierarchy where only a fewsuperboxes are visible on the screen.A *zoom* capability is provided to allow him to move into specific portions of the network, by replacing a group with its constituent boxes and groups. In this way, a browsing capability is provided for the Aurora diagram. oxes and groups have a tag, an argument list, a description of the Functionality, and, ultimately, a manual page.Users can teleport to specific places in an Aurora network by querying these attributes.Additionally, a user can place *bookmarks* in a network to allow him to return to places of interest. These capabilities give an Aurora user a mechanism to query the Aurora diagram.The user interface also allows monitors for arcs in the network to facilitate debugging as well as facilities for “single stepping” through a sequence of Aurora boxes.W e plan a graphical performance monitor as well as more sophisticated query capabilities.

4.3. SAMPLING DATA IN A STREAM

Finite-Population Sampling

Database sampling techniques have their roots in classical statistical methods for "finite-population sampling" (also called "survey sampling"). These latter methods are concerned with the problem of drawing inferences about a large finite population from a small random sample of population elements. The inferences usually take the form either of testing some hypothesis about the population—e.g., that a disproportionate number of smokers in the population suffer from emphysema—or estimating some parameters of the population— e.g., total income or average height. We focus primarily on the use of sampling for estimation of population parameters.

The simplest and most common sampling and estimation schemes require that the elements in a sample be "representative" of the elements in the population. The notion of simple random sampling (SRS) is one way of making this concept precise. To obtain an SRS of size k from a population of size n, a sample element is selected randomly and uniformly from among the n population elements, removed from the population, and added to the sample. This sampling step is repeated until k sample elements are obtained. The key property of an SRS scheme is that each of the (n k) possible subsets of k population elements is equally likely to be produced.

Other "representative" sampling schemes besides SRS are possible. An important example is simple random sampling with replacement (SRSWR).1 The SRSWR scheme is almost identical to SRS, except that each sampled element is returned to the population prior to the next random selection; thus a given population element can appear multiple times in the sample. When the sample size is very small with respect to the population size, the SRS and SRSWR schemes are almost indistinguishable, since the probability of sampling a given population element more than once is negligible. The mathematical theory of SRSWR is a bit simpler than that of SRS, so the former scheme is sometimes used as an approximation to the latter when analyzing estimation algorithms based on SRS. Other representative sampling schemes besides SRS and SRSWR include the "stratified" and "Bernoulli". As will become clear in the sequel, certain non-representative sampling methods are also useful in the data-stream setting. Of equal importance to sampling methods are techniques for estimating population parameters from sample data.

Suppose we wish to estimate the total income θ of a population of size n based on an SRS of size k, where k is much smaller than n. For this simple example, a natural estimator is obtained by scaling up the total income s of the individuals in the sample, $\hat{\theta} = (n/k)s$, e.g., if the sample comprises 1 % of the population, then scale up the total income of the sample by a factor of 100. For more complicated population parameters, such as the number of distinct ZIP codes in a population of magazine subscribers, the scale-up formula may be much less obvious. In general, the choice of estimation method is tightly coupled to the method used to obtain the underlying sample. Even for our simple example, it is important to realize that our estimate is random, since it depends on the particular sample obtained. For example, suppose (rather unrealistically) that our population consists of three individuals, say Smith, Abbas, and Raman, whose respective incomes are $10,000, $50,000, and 1Sometimes, to help distinguish between the two schemes more clearly, SRS is called simple random sampling without replacement.

Table 1 Possible scenarios, along with probabilities, for a sampling and estimation exercise

Sample	Sample income	Est. Pop. income	Scenario probability
{Smith, Abbas}	$60,000	$90,000	1/3
{Smith, Raman}	$1,010,000	$1,515,000	1/3
{Abbas, Raman}	$1,050,000	$1,575,000	1/3

$1,000,000. The total income for this population is $1,060,000. If we take an SRS of size k = 2—and hence estimate the income for the population as 1.5 times the income for the sampled individuals—then the outcome of our sampling and estimation exercise would follow one of the scenarios given in Table 1. Each of the scenarios is equally likely, and the expected value (also called the "mean value") of our estimate is computed as

expected value = (1/3) · (90,000)+(1/3) · (1,515,000)+(1/3) · (1,575,000)
= 1,060,000,

which is equal to the true answer. In general, it is important to evaluate the accuracy (degree of systematic error) and precision (degree of variability) of a sampling and estimation scheme. The bias, i.e., expected error, is a common measure of accuracy, and, for estimators with low bias, the standard error is a common measure of precision. The bias of our income estimator is 0 and the standard error is computed as the square root of the variance (expected squared deviation from the mean) of our estimator:

SE = #(1/3) · (90,000−1,060,000)2 +(1/3) · (1,515,000− 1,060,000)2
+(1/3) · (1,575,000− 1,060,000)2#1/2 ≈ 687,000.

For more complicated population parameters and their estimators, there are often no simple formulas for gauging accuracy and precision. In these cases, one can sometimes resort to techniques based on subsampling, that is, taking one or more random samples from the initial population sample.Well known subsampling techniques for estimating bias and standard error include the "jackknife" and "bootstrap" methods. In general, the accuracy and precision of a well designed sampling-based estimator should increase as the sample size increases.

Database Sampling

Although database sampling overlaps heavily with classical finite-population sampling, the former setting differs from the latter in a number of important respects.

• Scarce versus ubiquitous data. In the classical setting, samples are usually expensive to obtain and data is hard to come by, and so sample sizes tend to be small. In database sampling, the population size can be enormous (terabytes of data), and samples are relatively easy to collect, so that sample sizes can be relatively large. The emphasis in the database setting is on the sample as a flexible, lossy, compressed synopsis of the data that can be used to obtain quick approximate answers to user queries.

• Different sampling schemes. As a consequence of the complex storage formats and retrieval mechanisms that are characteristic of modern database systems, many sampling schemes that were unknown or of marginal interest in the classical setting are central to database sampling. For example, the classical literature pays relatively little attention to Bernoulli sampling schemes, but such schemes are very important for database sampling because they can be easily parallelized across data partitions. As another example, tuples in a relational database are typically retrieved from disk in units of pages or extents. This fact strongly influences the choice of sampling and estimation schemes, and indeed has led to the introduction of several novel methods. As a final example, estimates of the answer to an aggregation query involving select–project–join operations are often based on samples drawn individually from the input base relations, a situation that does not arise in the classical setting.

• No domain expertise. In the classical setting, sampling and estimation are often carried out by an expert statistician who has prior knowledge about the population being sampled. As a result, the classical literature is rife with sampling schemes that explicitly incorporate auxiliary information about the population, as well as "model-based" schemes in which the population is assumed to be a sample from a hypothesized "super-population" distribution. In contrast, database systems typically must view the population (i.e., the database) as a black box, and so cannot exploit these specialized techniques.

• Auxiliary synopses. In contrast to a classical statistician, a database designer often has the opportunity to scan each population element as it enters the system, and therefore has the opportunity to maintain auxiliary data synopses, such as an index of "outlier" values or other data

summaries, which can be used to increase the precision of sampling and estimation algorithms. If available, knowledge of the query workload can be used to guide synopsis creation.
Online-aggregation algorithms take, as input, streams of data generated by random scans of one or more (finite) relations, and produce continually-refined estimates of answers to aggregation queries over the relations, along with precision measures. The user aborts the query as soon as the running estimates are sufficiently precise; although the data stream is finite, query processing usually terminates long before the end of the stream is reached. Recent work on database sampling includes extensions of online aggregation methodology, application of bootstrapping ideas to facilitate approximate answering of very complex aggregation queries, and development of techniques for sampling-based discovery of correlations, functional dependencies, and other data relationships for purposes of query optimization and data integration.

Collective experience has shown that sampling can be a very powerful tool, provided that it is applied judiciously. In general, sampling is well suited to very quickly identifying pervasive patterns and properties of the data when a rough approximation suffices; for example, industrial-strength sampling-enhanced query engines can speed up some common decision-support queries by orders of magnitude. On the other hand, sampling is poorly suited for finding "needles in haystacks" or for producing highly precise estimates. The needle-in-haystack phenomenon appears in numerous guises. For example, precisely estimating the selectivity of a join that returns very few tuples is an extremely difficult task, since a random sample from the base relations will likely contain almost no elements of the join result. As another example, sampling can perform poorly when data values are highly skewed. For example, suppose we wish to estimate the average of the values in a data set that consists of 106 values equal to 1 and five values equal to 108. The five outlier values are the needles in the haystack: if, as is likely, these values are not included in the sample, then the sampling-based estimate of the average value will be low by orders of magnitude. Even when the data is relatively well behaved, some population parameters are inherently hard to estimate from a sample. One notoriously difficult parameter is the number of distinct values in a population. Problems arise both when there is skew in the data-value frequencies and when there are many data values, each appearing a small number of times. In the former scenario, those values that appear few times in the database are the needles in the haystack; in the latter scenario, the sample is likely to contain no duplicate values, in which case accurate assessment of a scale-up factor is impossible. Other challenging population parameters include the minimum or maximum data value. Researchers continue to develop new methods to deal with these problems, typically by exploiting auxiliary data synopses and workload information.

Sampling from Data Streams

Data-stream sampling problems require the application of many ideas and techniques from traditional database sampling, but also need significant new innovations, especially to handle queries over infinite-length streams. Indeed, the unbounded nature of streaming data represents a major departure from the traditional setting.We give a brief overview of the various stream-sampling techniques considered.

Our discussion centers around the problem of obtaining a sample from a window, i.e., a subinterval of the data stream, where the desired sample size is much smaller than the number of elements in the window. We draw an important distinction between a stationary window, whose endpoints are specified times or specified positions in the stream sequence, and a sliding window whose endpoints move forward as time progresses. Examples of the latter type of window include "the most recent n elements in the stream" and "elements that have arrived within the past hour." Sampling from a finite stream is a special case of sampling from a stationary window in which the window boundaries correspond to the first and last stream elements. When dealing with a stationary window, many traditional tools and techniques for database sampling can be directly brought to bear. In general, sampling from a sliding window is a much harder problem than sampling from a stationary window: in the former case, elements must be removed from the sample as they expire, and maintaining a sample of adequate size can be difficult.We also consider "generalized" windows

in which the stream consists of a sequence of transactions that insert and delete items into the window; a sliding window corresponds to the special case in which items are deleted in the same order that they are inserted.

Much attention has focused on SRS schemes because of the large body of existing theory and methods for inference from an SRS; we therefore discuss such schemes in detail. We also consider Bernoulli sampling schemes, as well as stratified schemes in which the window is divided into equal disjoint segments (the strata) and an SRS of fixed size is drawn from each stratum. Stratified sampling can be advantageous when the data stream exhibits significant autocorrelation, so that elements close together in the stream tend to have similar values. The foregoing schemes fall into the category of equal-probability sampling because each window element is equally likely to be included in the sample. For some applications it may be desirable to bias a sample toward more recent elements.

4.4. MINING DATA STREAMS AND MINING TIME

WINDOWING APPROACH TO DATA STREAM MINING

One of the main issues in the stream data mining is to find out a model which will suit the extraction process of the frequent item set from the streaming in data. There are three stream data processing model that are Landmark window, Damped window and Sliding window model. A transaction data stream is a sequence of incoming transactions and an excerpt of the stream is called a window. A window, W, can be either time-based or count-based, and either a landmark window or a sliding window. W is time-based if W consists of a sequence of fixed-length time units, where a variable number of transactions may arrive within each time unit. W is count-based if W is composed of a sequence of batches, where each batch consists of an equal number of transactions. W is a landmark window if W = (T1, T2, . . . , T); W is a sliding window if W = (TT−w+1, . . . , TT), where each Ti is a time unit or a batch, T1 and TT are the oldest and the current time unit or batch, and w is the number of time units or batches in the sliding window, depending on whether W is time-based or count-based. Note that a count-based window can also be captured by a time-based window by assuming that a uniform number of transactions arrive within each time unit.
The frequency of an item set, X, in W, denoted as freq(X), is the number of transactions in W support X. The support of X in W, denoted as sup(X), is defined as freq(X)/N, where N is the total number of transactions received in W. X is a Frequent Item set (FI) in W, if sup(X) $\geq \sigma$, where σ (0 $\leq \sigma \leq$ 1) is a user-specified minimum support threshold. X is a Frequent Maximal Item set (FMI) in W, if X is an FI in W and there exists no item set Y in W such that $X \subset Y$. X is a Frequent Closed Item set (FCI) in W, if X is an FI in W and there exists no item set Y in W such that $X \subset Y$ and freq(X) = freq(Y).

a) Landmark Window Concept
In this section we will discuss some of important land mark window algorithms. One of the algorithm proposed by Manku and Motwani is a lossy counting approximation algorithm. It will compute the approximate set of frequent item sets over the entire stream so far. In this algorithm the stream is divided into sequence of buckets. The lossy counting algorithm processes a batch of transactions arriving at a particular time. In this paper they are maintaining the item set, the frequency of item set and the error as the upper bound of the frequency of the item set. This algorithm uses three different modules, Buffer, Trie and Set Gen. The Buffer module keeps filling the available memory with the incoming transactions. This module frequently computes the frequency of every item in the current transactions and prune if it is less than N. The Trie module maintains set D, as a forest of prefix trees. The Trie forest as an array of tuples (X, freq(X), err (X), level) that correspond to the pre-order traversal of the forest, where the level of a node is the distance of the node from the root. The Trie array is maintained as a set of chunks. On updating the

Trie array, a new Trie array is created and chunks from the old Trie are freed as soon as they are not required.

All the item sets in the current batch having the support will be generated by the Set Gen module. The Apriori-like pruning[21] will help to avoid the generation of superset of an item set if the frequency less than β in the current batch. The Set Gen implemented with the help of Heap queue. Set Gen repeatedly processes the smallest item in Heap to generate a 1-itemset. If this 1-itemset is in Trie after the Add Entry or the Update Entry operation is utilized, Set Gen is recursively invoked with a new Heap created out of the items that follow the smallest items in the same transactions. During each call of Set Gen, qualified old item sets are copied to the new Trie array according to their orders in the old Trie array, while at the same time new item sets are added to the new Trie array in lexicographic order. When the recursive call returns, the smallest entry in Heap is removed and the recursive process continues with the next smallest item in Heap.

The quality of the approximation mining results by using the relaxed minimum support threshold leads to the extra usage of memory and the processing power. That is, the smaller relaxed minimum support leads to increase of number of sub-FIs generated, so the increase of memory and the extra usage of processing power. , if approaches σ, more false-positive answers will be included in the result, since all sub-FIs whose computed frequency is at least $(\sigma -)N \approx 0$ are displayed while the computed frequency of the sub-FIs can be less than their actual frequency by as much as σN. The same problem is in other mining algorithms [21, 22, 23, 24, 13, 4] that use a relaxed minimum support threshold to control the accuracy of the mining result. ∈∈ ∈

One of the algorithm called DSM-FI developed by Li[13], is to mine an approximate set of FIs over the entire history of the stream. This algorithm is used a prefix-tree based in memory data structure. DSM-FI is also using the relaxed minimum support threshold and all the generated FIs are stored in the IsFI-forest. The DSM-FI consists of Header Table(HT) and Sub-Frequent Itemsets tree(SFI-tree). For every unique item in the set of sub-FIs it inserts an entry with frequency, batch id and head link, it increments otherwise. The DSM-FI frequently prunes the items that are not satisfied the minimum support.

One of the approximation algorithm developed by Lee[4] used the compressed prefix tree structure called CP-tree. The structure of the CP-tree is described as follows. Let D be the prefix tree used in estDec. Given a merging gap threshold δ, where $0 \leq \delta \leq 1$, if all the itemsets stored in a subtree S of D satisfy the following equation, then S is compressed into a node in the CP-tree.

$$\frac{freq_T(X) - freq_T(Y)}{N_T} \leq \delta$$

Where X is the root of S and Y is an item set in S. Assume S is compressed into a node v in the CP-tree. The node v consists of the following four fields: item-list, parent-list, freqTmax and freqTmin where v.item-list is a list of items which are the labels of the nodes in S, v. parent-list is a list of locations (in the CP-tree) of the parents of each node in S, v. freqTmax is the frequency of the root of S and freqTmin is the frequency of the right-most leaf of S.

The use of the CP-tree results in the reduction of memory consumption, which is important in mining data streams. The CP-tree can also be used to mine the FIs, however, the error rate of the computed frequency of the FIs, which is estimated from freqTmin and freqTmax, will be further increased. Thus, the CP-tree is more suitable for mining FMIs.

b) Sliding Window Concept

The sliding window model processes only the items in the window and maintains only the frequent item sets. The size of the sliding window can be decided according to the applications and the system resources. The recently generated transactions in the window will influence the mining result of the sliding windowing, otherwise all the items in the window to be maintained. The size of the sliding window may vary depends up on the applications it may use. In this section we will discuss some of the important windowing approaches for stream mining.

An in memory prefix tree based algorithm following the windowing approach to

incrementally update the set of frequent closed item sets over the sliding window . The data structure used for the algorithm is called as Closed Enumeration Tree (CET) to maintain the dynamically selected set of item set over the sliding window. This algorithm will compute the exact set of frequent closed item sets over the sliding window. The updation will be for each incoming transaction but not enough to handle the handle the high speed streams.

One another notable algorithm in the windowing concept is estWin[3]. This algorithm maintains the frequent item sets over a sliding window. The data structure used to maintain the item sets is prefix tree. The prefix tree holds three parameters for each items set in the tree, that are frequency of x in current window before x is inserting in the tree, that is freq(x). The second is an upper bound for the frequency of x in the current window before x is inserted in the tree, err(x). The third is the ID of the transaction being processed, tid(x).b. The item set in the tree will be pruned along with all supersets of the item set, we prune the item set X and the supersets if $tid(X) \leq tid1$ and $freq(X) < \lceil N \rceil$, or (2) $tid(X) > tid1$ and $freq(X) < \lceil (N - (tid(X) - tid1)) \rceil$. The expression $tid(X) > tid1$ means that X is inserted into D at some transaction that arrived within the current sliding window and hence the expression $(N-(tid(X)-tid1))$returns the number of transactions that arrived within the current window since the arrival of the transaction having the ID tid(X). We note that X itself is not pruned if it is a 1-itemset, since estWin estimates the maximum frequency error of an itemset based on the computed frequency of its subsets [84] and thus the frequency of a 1-itemset cannot be estimated again if it is deleted. ∈∈

c) Damped Window Concept

The estDec algorithm proposed to reduce the effect of the old transactions on the stream mining result. They have used a decay rate to reduce the effect of the old transactions and the resulted frequent item sets are called recent frequent Item sets. The algorithm, for maintaining recent FIs is an approximate algorithm that adopts the mechanism to estimate the frequency of the item sets.

The use of a decay rate diminishes the effect of the old and obsolete information of a data stream on the mining result. However, estimating the frequency of an item set from the frequency of its subsets can produce a large error and the error may propagate all the way from the 2-subsets to the n-supersets, while the upper bound is too loose. Thus, it is difficult to formulate an error bound on the computed frequency of the resulting item sets and a large number of false-positive results will be returned, since the computed frequency of an item set may be much larger than its actual frequency. Moreover, the update for each incoming transaction (instead of a batch) may not be able to handle high-speed streams.

Another approximation algorithm uses a tilted time window model . In this frequency FIs are kept in different time granularities such as last one hour, last two hours, last four hours and so on. The data structure used in this algorithm is called FP-Stream. There are two components in the FP-Stream which are pattern tree based prefix tree and tilted time window which is at the end node of the path. The pattern tree can be constructed using the FP-tree algorithm. The tilted time window guarantees that the granularity error is at most T/2, where T is the time units.

The updation of the frequency record will be done by shifting the recent records to merge with the older records. To reduce the number of frequency records in the tilted-time windows, the old frequency records of an item set, X, are pruned as follows. Let freqj(X) be the computed frequency of X over a time unit Tj and Nj be the number of transactions received within Tj , where $1 \leq j \leq \tau$. For some m, where $1 \leq m \leq \tau$, the frequency records freq1(X), . . . ,freqm(X) are pruned if the following condition holds:

$\exists n \leq \tau, \forall i, 1 \leq i \leq n$, $freqi(X) < \sigma Ni$ and

$\forall l, 1 \leq l \leq m \leq n$, $\Sigma-(x) < \Sigma$

The FP-stream mining algorithm computes a set of sub-FIs at the relaxed minimum support threshold, , over each batch of incoming transactions by using the FI mining algorithm, FP-growth [25]. For each sub-FI X obtained, FP-streaming inserts X into the FP-stream if X is not in the FP-stream. If X is already in the FP-stream, then the computed frequency of X over the current batch is added to its tilted-time window. Next, pruning is performed on the tilted-time window of X and if

the window becomes empty, FP-growth stops mining supersets of X by the Apriori property. After all sub-FIs mined by FP-growth are updated in the FP-stream, the FP-streaming scans the FP-stream and, for each item set X visited, if X is not updated by the current batch of transactions, the most recent frequency in X's tilted-time window is recorded as 0. Pruning is then performed on X. If the tilted-time window of some item set visited is empty (as a result of pruning), the item set is also pruned from the FP-stream. ∈

The tilted-time window model allows us to answer more expressive time-sensitive queries, at the expense of some frequency record kept for each item set. The tilted-time window also places greater importance on recent data than on old data as does the sliding window model; however, it does not lose the information in the historical data completely. A drawback of the approach is that the FP-stream can become very large over time and updating and scanning such a large structure may degrade the mining throughput.

4.5. SERIES DATA

A time series is a series of data points indexed (or listed or graphed) in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data. Examples of time series are heights of ocean tides, counts of sunspots, and the daily closing value of the Dow Jones Industrial Average.

Time series are very frequently plotted via line charts. Time series are used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather forecasting, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering, and largely in any domain of applied science and engineering which involves temporal measurements.

Time series analysis comprises methods for analyzing time series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the use of a model to predict future values based on previously observed values. While regression analysis is often employed in such a way as to test theories that the current values of one or more independent time series affect the current value of another time series, this type of analysis of time series is not called "time series analysis", which focuses on comparing values of a single time series or multiple dependent time series at different points in time. Interrupted time series analysis is the analysis of interventions on a single time series

Time series data have a natural temporal ordering. This makes time series analysis distinct from cross-sectional studies, in which there is no natural ordering of the observations (e.g. explaining people's wages by reference to their respective education levels, where the individuals' data could be entered in any order). Time series analysis is also distinct from spatial data analysis where the observations typically relate to geographical locations (e.g. accounting for house prices by the location as well as the intrinsic characteristics of the houses). A stochastic model for a time series will generally reflect the fact that observations close together in time will be more closely related than observations further apart. In addition, time series models will often make use of the natural one-way ordering of time so that values for a given period will be expressed as deriving in some way from past values, rather than from future values.

Time series analysis can be applied to real-valued, continuous data, discrete numeric data, or discrete symbolic data (i.e. sequences of characters, such as letters and words in the English language).

Methods for analysis

Methods for time series analysis may be divided into two classes: frequency-domain methods and time-domain methods. The former include spectral analysis and wavelet analysis; the latter include auto-correlation and cross-correlation analysis. In the time domain, correlation and analysis can be made in a filter-like manner using scaled correlation, thereby mitigating the need to operate in the frequency domain.

Additionally, time series analysis techniques may be divided into parametric and non-

parametric methods. The parametric approaches assume that the underlying stationary stochastic process has a certain structure which can be described using a small number of parameters (for example, using an autoregressive or moving average model). In these approaches, the task is to estimate the parameters of the model that describes the stochastic process. By contrast, non-parametric approaches explicitly estimate the covariance or the spectrum of the process without assuming that the process has any particular structure.

Methods of time series analysis may also be divided into linear and non-linear, and univariate and multivariate.

Panel data

A time series is one type of panel data. Panel data is the general class, a multidimensional data set, whereas a time series data set is a one-dimensional panel (as is a cross-sectional dataset). A data set may exhibit characteristics of both panel data and time series data. One way to tell is to ask what makes one data record unique from the other records. If the answer is the time data field, then this is a time series data set candidate. If determining a unique record requires a time data field and an additional identifier which is unrelated to time (student ID, stock symbol, country code), then it is panel data candidate. If the differentiation lies on the non-time identifier, then the data set is a cross-sectional data set candidate.

Analysis

There are several types of motivation and data analysis available for time series which are appropriate for different purposes and etc.

Motivation

In the context of statistics, econometrics, quantitative finance, seismology, meteorology, and geophysics the primary goal of time series analysis is forecasting. In the context of signal processing, control engineering and communication engineering it is used for signal detection and estimation, while in the context of data mining, pattern recognition and machine learning time series analysis can be used for clustering, classification, query by content, anomaly detection as well as forecasting.

Exploratory analysis

The clearest way to examine a regular time series manually is with a line chart such as the one shown for tuberculosis in the United States, made with a spreadsheet program. The number of cases was standardized to a rate per 100,000 and the percent change per year in this rate was calculated. The nearly steadily dropping line shows that the TB incidence was decreasing in most years, but the percent change in this rate varied by as much as +/- 10%, with 'surges' in 1975 and around the early 1990s. The use of both vertical axes allows the comparison of two time series in one graphic.

Other techniques include:

Autocorrelation analysis to examine serial dependence

Spectral analysis to examine cyclic behavior which need not be related to seasonality. For example, sun spot activity varies over 11 year cycles. Other common examples include celestial phenomena, weather patterns, neural activity, commodity prices, and economic activity.

Separation into components representing trend, seasonality, slow and fast variation, and cyclical irregularity: see trend estimation and decomposition of time series

Curve fitting

Curve fitting is the process of constructing a curve, or mathematical function, that has the best fit to a series of data points, possibly subject to constraints. Curve fitting can involve either interpolation, where an exact fit to the data is required, or smoothing, in which a "smooth" function

is constructed that approximately fits the data. A related topic is regression analysis, which focuses more on questions of statistical inference such as how much uncertainty is present in a curve that is fit to data observed with random errors. Fitted curves can be used as an aid for data visualization, to infer values of a function where no data are available, and to summarize the relationships among two or more variables. Extrapolation refers to the use of a fitted curve beyond the range of the observed data, and is subject to a degree of uncertainty since it may reflect the method used to construct the curve as much as it reflects the observed data.

The construction of economic time series involves the estimation of some components for some dates by interpolation between values ("benchmarks") for earlier and later dates. Interpolation is estimation of an unknown quantity between two known quantities (historical data), or drawing conclusions about missing information from the available information ("reading between the lines"). Interpolation is useful where the data surrounding the missing data is available and its trend, seasonality, and longer-term cycles are known. This is often done by using a related series known for all relevant dates. Alternatively polynomial interpolation or spline interpolation is used where piecewise polynomial functions are fit into time intervals such that they fit smoothly together. A different problem which is closely related to interpolation is the approximation of a complicated function by a simple function (also called regression).The main difference between regression and interpolation is that polynomial regression gives a single polynomial that models the entire data set. Spline interpolation, however, yield a piecewise continuous function composed of many polynomials to model the data set.

Extrapolation is the process of estimating, beyond the original observation range, the value of a variable on the basis of its relationship with another variable. It is similar to interpolation, which produces estimates between known observations, but extrapolation is subject to greater uncertainty and a higher risk of producing meaningless results.

Function approximation

In general, a function approximation problem asks us to select a function among a well-defined class that closely matches ("approximates") a target function in a task-specific way. One can distinguish two major classes of function approximation problems: First, for known target functions approximation theory is the branch of numerical analysis that investigates how certain known functions (for example, special functions) can be approximated by a specific class of functions (for example, polynomials or rational functions) that often have desirable properties (inexpensive computation, continuity, integral and limit values, etc.).

Second, the target function, call it g, may be unknown; instead of an explicit formula, only a set of points (a time series) of the form (x, g(x)) is provided. Depending on the structure of the domain and codomain of g, several techniques for approximating g may be applicable. For example, if g is an operation on the real numbers, techniques of interpolation, extrapolation, regression analysis, and curve fitting can be used. If the codomain (range or target set) of g is a finite set, one is dealing with a classification problem instead. A related problem of online time series approximation is to summarize the data in one-pass and construct an approximate representation that can support a variety of time series queries with bounds on worst-case error.

To some extent the different problems (regression, classification, fitness approximation) have received a unified treatment in statistical learning theory, where they are viewed as supervised learning problems.

Prediction and forecasting

In statistics, prediction is a part of statistical inference. One particular approach to such inference is known as predictive inference, but the prediction can be undertaken within any of the several approaches to statistical inference. Indeed, one description of statistics is that it provides a means of transferring knowledge about a sample of a population to the whole population, and to other related populations, which is not necessarily the same as prediction over time. When information is transferred across time, often to specific points in time, the process is known as

forecasting.

Fully formed statistical models for stochastic simulation purposes, so as to generate alternative versions of the time series, representing what might happen over non-specific time-periods in the future

Simple or fully formed statistical models to describe the likely outcome of the time series in the immediate future, given knowledge of the most recent outcomes (forecasting).

Forecasting on time series is usually done using automated statistical software packages and programming languages, such as R, S, SAS, SPSS, Minitab, pandas (Python) and many others.

Classification

Assigning time series pattern to a specific category, for example identify a word based on series of hand movements in sign language

Signal estimation

This approach is based on harmonic analysis and filtering of signals in the frequency domain using the Fourier transform, and spectral density estimation, the development of which was significantly accelerated during World War II by mathematician Norbert Wiener, electrical engineers Rudolf E. Kálmán, Dennis Gabor and others for filtering signals from noise and predicting signal values at a certain point in time. See Kalman filter, Estimation theory, and Digital signal processing

Segmentation

Splitting a time-series into a sequence of segments. It is often the case that a time-series can be represented as a sequence of individual segments, each with its own characteristic properties. For example, the audio signal from a conference call can be partitioned into pieces corresponding to the times during which each person was speaking. In time-series segmentation, the goal is to identify the segment boundary points in the time-series, and to characterize the dynamical properties associated with each segment. One can approach this problem using change-point detection, or by modeling the time-series as a more sophisticated system, such as a Markov jump linear system.

Models

Models for time series data can have many forms and represent different stochastic processes. When modeling variations in the level of a process, three broad classes of practical importance are the autoregressive (AR) models, the integrated (I) models, and the moving average (MA) models. These three classes depend linearly on previous data points.Combinations of these ideas produce autoregressive moving average (ARMA) and autoregressive integrated moving average (ARIMA) models. The autoregressive fractionally integrated moving average (ARFIMA) model generalizes the former three. Extensions of these classes to deal with vector-valued data are available under the heading of multivariate time-series models and sometimes the preceding acronyms are extended by including an initial "V" for "vector", as in VAR for vector autoregression. An additional set of extensions of these models is available for use where the observed time-series is driven by some "forcing" time-series (which may not have a causal effect on the observed series): the distinction from the multivariate case is that the forcing series may be deterministic or under the experimenter's control. For these models, the acronyms are extended with a final "X" for "exogenous".

Non-linear dependence of the level of a series on previous data points is of interest, partly because of the possibility of producing a chaotic time series. However, more importantly, empirical investigations can indicate the advantage of using predictions derived from non-linear models, over those from linear models, as for example in nonlinear autoregressive exogenous models. Further references on nonlinear time series analysis.

Among other types of non-linear time series models, there are models to represent the changes of variance over time (heteroskedasticity). These models represent autoregressive

conditional heteroskedasticity (ARCH) and the collection comprises a wide variety of representation (GARCH, TARCH, EGARCH, FIGARCH, CGARCH, etc.). Here changes in variability are related to, or predicted by, recent past values of the observed series. This is in contrast to other possible representations of locally varying variability, where the variability might be modelled as being driven by a separate time-varying process, as in a doubly stochastic model.

In recent work on model-free analyses, wavelet transform based methods (for example locally stationary wavelets and wavelet decomposed neural networks) have gained favor. Multiscale (often referred to as multiresolution) techniques decompose a given time series, attempting to illustrate time dependence at multiple scales. See also Markov switching multifractal (MSMF) techniques for modeling volatility evolution.

A Hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states. An HMM can be considered as the simplest dynamic Bayesian network. HMM models are widely used in speech recognition, for translating a time series of spoken words into text.

Notation

A number of different notations are in use for time-series analysis. A common notation specifying a time series X that is indexed by the natural numbers is written

$X = \{X1, X2, ...\}$.

Another common notation is

$Y = \{Yt: t \in T\}$,

where T is the index set.

Conditions

There are two sets of conditions under which much of the theory is built:

Stationary process

Ergodic process

However, ideas of stationarity must be expanded to consider two important ideas: strict stationarity and second-order stationarity. Both models and applications can be developed under each of these conditions, although the models in the latter case might be considered as only partly specified.

In addition, time-series analysis can be applied where the series are seasonally stationary or non-stationary. Situations where the amplitudes of frequency components change with time can be dealt with in time-frequency analysis which makes use of a time–frequency representation of a time-series or signal.

4.6. REAL TIME ANALYTICS PLATFORM (RTAP) APPLICATIONS

Real time Analytics Platform (RTAP) Applications

Real Time Analytics Platform (RTAP) analyzes data, correlates and predicts outcomes on a real time basis. The platform enables enterprises to track things in real time on a worldwide basis and helps in timely decision making. This platform provides us to build a range of powerful analytic applications. The platform has two key functions: they manage stored data and execute analytics program against it.

Why we need RTAP?

RTAP addresses the following issues in the traditional or existing RDBMS system

- Server based licensing is too expensive to use large DB servers
- Slow processing speed
- Little support tools for data extraction outside data warehouse
- Copying large datasets into system is too slow
- Workload differences among jobs

 Data kept in files and folder, managing them are difficult

Analytic platform combines tools for creating analyses with an engine to execute them, a DBMS to keep and manage them for ongoing use and mechanism for acquiring and preparing data that are not already stored. The components of the platform are depicted in the figure.

The data can be collected from multiple data sources and feed through the data integration process. The data can be captured and transformed and loaded into analytic database management system (ADBMS). This ADBMS has separate data store to mange data. It also has the provision for creating functions and procedures to operate on the data. Models can be created for analysis in the ADBMS itself. Analytic applications can make use of the data in the ADBMS and apply the algorithms on it.

The application has the following facilities

 Ad-hoc reporting

 Model building

 Statistical Analysis

 Predictive Analysis

 Data visualization

ADBMS has the following desirable features for data analytics

 Use of proprietary hardware

 Hardware sharing model for processing and data through MPP (Massive Parallel Processing)

 Storage format (row and column manner) and smart data management

 SQL support and NoSQL support

 Programming extensibility and more cores, threads which yield more processing power

 Deployment model

 Infiniband -speed network

Applications

Social Media Analytics

Social Media is the modern way of communication and networking. It is a growing and widely accepted way of interaction these days and connects billions of people on a real time basis.

 Fan page analysis – face book and twitter

 Tweeter analysis on followers, locations, time to tweet, interest to influence

 Measure customer service metrics on twitter

 Social media impacts on website performance

Business Analytics

Business analytics focuses on developing new insights and understanding of business performance based on data and statistical methods. It gives critical information about supply and

demand of business/product's viability in the marketplace.
 Goal tracking and returning customers

 trendlines

 Brand influence and reputation

 Combines online marketing and e-commerce data

Customer analytics
 Customer profiling

 Customer segmentation based on behaviour

 Customer retention by increasing lifetime value

Applications: Google, IBM, SAS, WEKA analytic tools

Web Analytics

It is the process of collecting, analyzing and reporting of web data for the purpose of understanding and optimizing web usage.
 On-site analytics (No. of users visited, no. of current users and actions, user locations etc…)
 Logfile analysis
 Click analytics
 Customer life cycle analytics
 Tracking web traffic

Applications: clicky, shinystat, statcounter, site meters etc..

4.7. CASE STUDIES - REAL TIME SENTIMENT ANALYSIS, STOCK MARKET PREDICTIONS.

Introduction

Historically, stock market movements have been highly unpredictable. With the advent of technological advances over the past two decades, nancial institutions and researchers have developed computerized mathematical models to maximize their returns while minimizing their risk. One recent model by Johan Bollen involves analyzing the public's emotional states, represented by Twitter feeds, in order to predict the market. The state of the art in sentiment analysis suggests there are 6 important mood states that enable the prediction of mood in the general public. The prediction of mood uses the sentiment word lists obtained in various sources where general state of mood can be found using such word list or emotion tokens. With the number of tweets posted on Twitter, it is believed that the general state of mood can be predicted with certain statistical signicance.

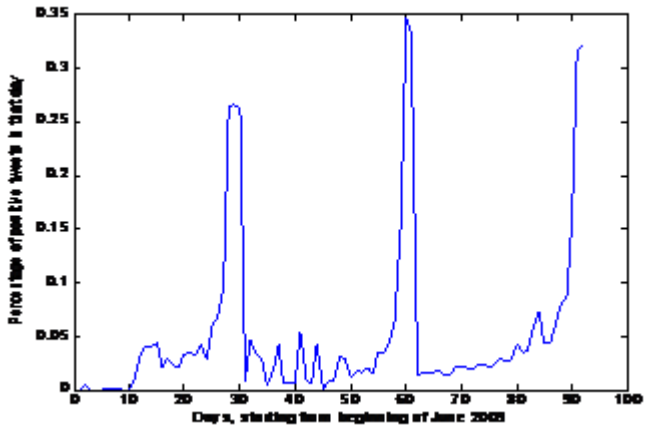
According to Bollen's paper, Twitter sentiment is correlated with the market, preceding it by a few days. Specically, the Google Prole of Mood States' (GPOMS) `calm' state proved to be a reliable predictor of the market. Due to the proprietary nature of the GPOMS algorithm, we wish to see if a simpler method could provide similar results, while still being able to make accurate enough predictions to be protable.

Sentiment Analysis

We begin our sentiment analysis by applying Alex Davies' word list in order to see if a simple approach is sucient enough to correlate to market movement. For this, we use a pre-

generated word list of roughly ve thousand common words along with log probabilities of `happy' or `sad' associated with the respective words. The process works as follows. First, each tweet is tokenized into a word list. The parsing algorithm separates the tweets using whitespace and punctuation, while accounting for common syntax found in tweets, such as URLs and emoticons. Next, we look up each token's log-probability in the word list; as the word list is not comprehensive, we choose to ignore words that do not appear in the list. The log probabilities of each token was simply added to determine the probability of `happy' and `sad' for the entire tweet. These were then averaged per day to obtain a daily sentiment value.

As expected, this method resulted in highly uncorrelated data (with correlation coecients of almost zero). We tried to improve this by using a more comprehensive and accurate dictionary for positive and negative sentiments. Specically, we swapped our initial word list with a sentiment score list we generated using SentiWordNet, which consisted of over 400 thousand words. Since this list considers relationships between each word and includes multi-word expressions, it provided better results. We also tried representing the daily sentiment value in a dierent way - instead of averaging the probabilities of each tweet, we counted the frequency of `happy' tweets (such as using a threshold probability of above 0.5 for happy) and represented this as a percentage of all tweets for that day. While this did not improve the output's correlation with stock market data, it did provide us with more insight into our Twitter data. For example, we see a spike in the percentage `happy' tweets toward the end of each month (Figure 1).



We did not news events which could have caused these spikes; however, upon investigating the source of Twitter data, we found that it had been pre-ltered for a previous research project (i.e. there may be some bias in what we assumed to be raw Twitter data). Due to a lack of access to better Twitter data, we conclude that using the frequency of happy tweets is not a reliable indicator of sentiment for our application and revert to our averaging method.

Constructing the Model

In this section, we discuss the construction of our model, from choosing an appropriate algorithm to finding a suitable set of features, and provide justication for these decisions.

The Algorithm

We chose to model the data using a linear regression. This decision was motivated by several factors:

Speed - A fast, ecient algorithm was one of our original specications. This is a must when working with massive amounts of data in real time, as is the case in the stock market.

Regression - We sought to be able to make investment decisions not only on direction of market movement, but also to quantify this movement. A simple classier was insucient for this; we required a regressor.

Accurate - Naturally, we needed an algorithm that would model the data as accurately as possible. Since our data is, by its nature, very noisy, we chose a simple model to avoid high variance.

Features

The backbone of our algorithm was, of course, Twitter sentiment data. As such, we designed several features that correspond to these sentiment values at various time-delays to the present. Training in one-dimensional feature space using only this data, we found that the best results were obtained when the Twitter data predated the market by 3 days. Using k-fold cross-validation to quantify our accuracy, we observed that this model was able to make predictions with approximately 60% accuracy, a modest improvement over no information (50% accuracy), but we wanted to see if we could do better.

We designed 2 more classes of features to try: one modeling the change in price of the market each day at various time-delays, the other modeling the total change in price of the market over the past n days. To help us choose a good set of features, we applied a feature selection algorithm using forward search to the problem. From this, we learned that the `change in price 3 days ago' feature improved our previous model to one with approximately 64% accuracy.

Further tests indicated that several of the other features are also relevant, however, due to relatively small amount of training data (72 days or fewer), training in higher-dimensional feature spaces yielded worse results in practice. Nonetheless, with the availability of more training data, a more complex and diverse set of features could further improve accuracy. We were able to achieve, using nearly all of our available data to train (infeasible for portfolio simulation, see next section), classication accuracy as high as 70%.

Testing the Model

We have built a model for predicting changes in the stock market price from day to day. We have identify the accuracy-maximizing set of features and trained our model on these features. Now we must put it to the test using real-world data to determine if it is protable. We develop 2 different investment strategies based on predictions from our model, apply them over some time period, report on the results, and compare them to 2 benchmark investment strategies.

Our Investment Strategies

Classication - The simpler of our 2 strategies considers only the predicted direction of market movement. That is, we look only at the sign of the anticipated change in price. If it is positive, we buy as many shares as possible with our current funds. Otherwise, we buy no shares, and simply sit on the money until the following day when we reevaluate the situation.

Regression - With this more complicated strategy, we seek to exploit knowledge of how much the market will change, rather than simply the direction it will shift. This allows us to base how much we invest on how certain we are of our prediction. There are countless such strategies that one could propose, we chose the following based on observations of good performance:

$$invest = \begin{cases} 100\% & \text{if } .05\% < (predicted\ \%\ change) \\ 25\% & \text{if } -.1\% \leqslant (predicted\ \%\ change) \leqslant .05\% \\ 0\% & \text{if } (predicted\ \%\ change) < -.1\% \end{cases}$$

Here, invest is the percent of our funds we use to buy stock and predicted % change q is computed by dividing the predicted change in the market tomorrow by the price today.

The Benchmark Investment Strategies

Default - This strategy uses no information about the market, and will simply buy as many shares as possible each day.

Maximal - This strategy assumes perfect knowledge about future stock prices. We will invest all

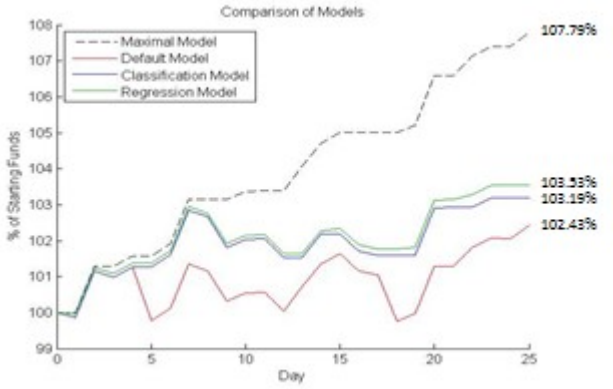
available funds when we know the market will go up the following day, and invest no funds when we know the market will go down. This strategy is, of course, impossible to execute in reality, and is only being used to quantify the prots from an ideal strategy.

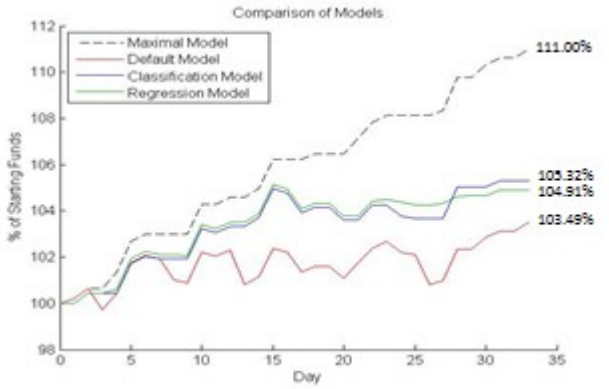
Simulation

We start with exactly enough money to buy 50 shares of stock on the rst day. Note that since we output results as percentages of starting money, they do not depend on this value, and as such it is chosen arbitrarily. At the start of each day, we make a prediction and invest according to some strategy. At the end of the day, we sell all shares at closing price and put the money in this bank. This is done so that any gains or losses can future gains or losses by virtue of being able to purchase more or less stock at every time step.

Results

We ran the simulation for each investment strategy, as described above, on 2 dierent time intervals. The results are shown below:





In the gure on the left, we trained on about 3 4 of the data (72 days) and simulated on about 1 4 of the data (25 days). In the gure on the right, we trained on about 2 3 of the data (64 days) and simulated on about 1 3 of the data (33 days). We immediately see that both of our strategies fare better than the default strategy in both simulations.

Note, however, that the regression strategy is more protable in the rst simulation while the classification strategy is more protable in the second simulation. We observe that on the simulation in which the model was given less training data (figure on the right), on day 27, our regression strategy opted to invest only 25% of funds that day because it perceived gains as being uncertain. This did not happen on the corresponding day in the first simulation (with more training data). Indeed, with less data to train on, imprecision in our model resulted in a poor investment decision when using the more complex regression strategy. In general, the classification strategy tends to be more consistent, while the regression strategy, though theoretically more protable, is also more sensitive to noise in the model.